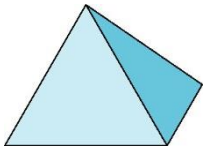
Lecture 6

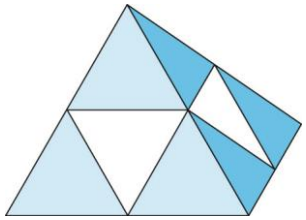
The 3D Gasket

How to construct the 3D Gasket

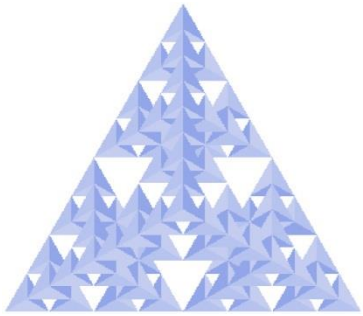
We start with a tetrahedron



We thin find and connect the six midpoints on the edges of the tetrahedron. This gives four smaller tetrahedrons, one for each original vertex and a cutout area



Recursively continue subdivide each resulting tetrahedron for a specified numbers and finally draw the resulting small ones.



3D Gasket program

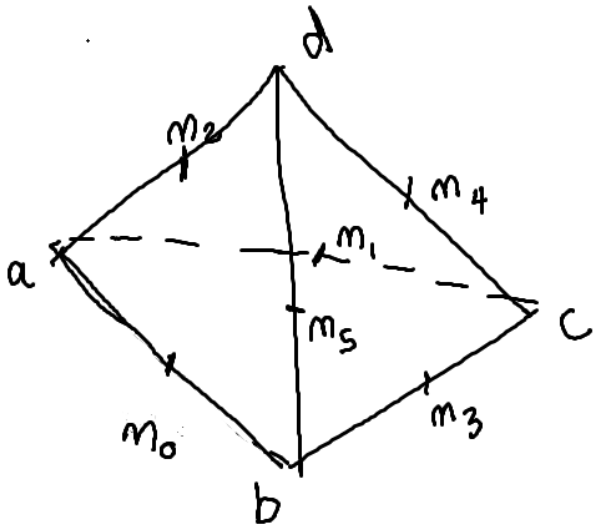
We need a function to draw the resulting tetrahedron triangles

We need a function that uses the above function to draw a tetrahedron

```
void triangle(GLfloat *va, GLfloat *vb, GLfloat *vc)
{
        glVertex3fv(va);
        glVertex3fv(vb);
        glVertex3fv(vc);
}

void tetra(GLfloat *a, GLfloat *b, GLfloat *c, GLfloat *d)
{
    glColor3fv(colors[0]);
    triangle(a, b, c);
    glColor3fv(colors[1]);
    triangle(a, c, d);
    glColor3fv(colors[2]);
    triangle(a, d, b);
    glColor3fv(colors[3]);
    triangle(b, d, c);
}
```

3D Gasket program: division of the tetrahedron

```
void divide_tetra(GLfloat *a, GLfloat *b, GLfloat *c, GLfloat *d, int m)
{

    GLfloat mid[6][3];
    int j;
    if(m>0)
    {
        /* compute six midpoints */

        for(j=0; j<3; j++) mid[0][j]=(a[j]+b[j])/2;
        for(j=0; j<3; j++) mid[1][j]=(a[j]+c[j])/2;
        for(j=0; j<3; j++) mid[2][j]=(a[j]+d[j])/2;
        for(j=0; j<3; j++) mid[3][j]=(b[j]+c[j])/2;
        for(j=0; j<3; j++) mid[4][j]=(c[j]+d[j])/2;
        for(j=0; j<3; j++) mid[5][j]=(b[j]+d[j])/2;

        /* create 4 tetrahedrons by subdivision */

        divide_tetra(a, mid[0], mid[1], mid[2], m-1);
        divide_tetra(mid[0], b, mid[3], mid[5], m-1);
        divide_tetra(mid[1], mid[3], c, mid[4], m-1);
        divide_tetra(mid[2], mid[4], d, mid[5], m-1);

    }
    else(tetra(a,b,c,d)); /* draw tetrahedron at end of recursion */
}
```



3D Gasket program: integrating the parts

```
GLfloat v[4][3]={{0.0, 0.0, 1.0}, {0.0, 0.942809, -0.33333},
      {-0.816497, -0.471405, -0.333333}, {0.816497, -0.471405, -0.333333}};

GLfloat colors[4][3] = {{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0},
                        {0.0, 0.0, 1.0}, {0.0, 0.0, 0.0}};
```

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    divide_tetra(v[0], v[1], v[2], v[3], n);
    glEnd();
    glFlush();
}
```

Hidden surface removal

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

GL_DEPTH_BUFFER_BIT makes OpenGL clear the z buffer in the frame buffer, when clearing the graphics

```
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
```

GL_DEPTH makes OpenGL to enable the z or depth buffer (a layer in the frame buffer)

```
glEnable(GL_DEPTH_TEST)
```

GL_DEPTH_TEST Make OpenGL enable the hidden surface removal algorithm that uses depth (z) to remove hidden surfaces

```
int main(int argc, char **argv)
{
    n=atoi(argv[1]); /* or enter number of subdivision steps here */
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("3D Gasket");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
    glEnable(GL_DEPTH_TEST);
    glClearColor (1.0, 1.0, 1.0, 1.0);
    glutMainLoop();
}
```

Drawing Implicit and explicit functions

$Implicit\ function\colon\ g(x,y)=0$ $Explicit\ function\colon\ Y=h(x)$

$Emplicit\ example\ (circle\ centered\ at\ the\ origin)\colon\ x^2+y^2-1=0$

$Emplicit\ example\ (Ovals\ of\ Cassini)\colon (x^2+y^2+a^2)-4a^2x^2-b^4=0$

$Explicit\ (Sine\ function)\colon y=\sin(x)$

Explicit functions are already separable. Then we can substitute in the function with the input and draw
In the case of Implicit function: The function may generate no, one or many curves depending on the parameter and the equation (example 2, for example, may generate no, one or two curves depending on the values of a and b

Explicit function: General form

$General\colon Z = f(x, y)$ $Contour\colon c = f(x, y)\ or\ \ f(x, y) - c = 0$

In this general form, instead of having no, one or many curve for the case z=0, we will have all these possibilities for every possible value of z. The curves generated for a given value of z=c are called the counters for z=c.

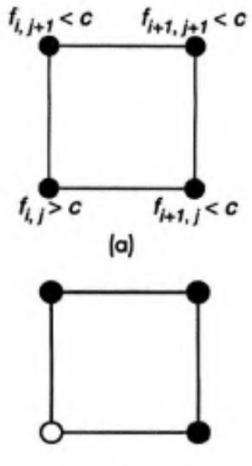
To find the contour for c, two ways could be taken:

- Solving the implicit equation: $f(x, y) - c = 0$, for the x , y values (difficult or impossible, no systematic way to go on)
- Sampling the region of interest in both x and y, study the function values at these samples to see if the contour pass by them or not. If it passes draw it, otherwise don't draw. (there is a systematic way to go on)

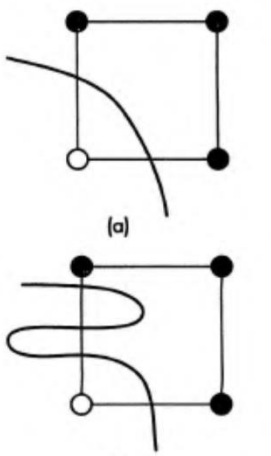
The second way is an imitation to sampling an object using devices such as laser or satellite. Note also that the second way only approximate the contours

Approximating a contour: Marching squares

We start by a grid of x and y (points where samples will be tested). For each cell of the grid, we see if the contour pass by the cell or not. If it passes, how it passes (approximation of the intersection point(s) of the contour with the cell edges. Finally, we connect these points of intersection using line segment to get an approximation for the contour. The following is an example for a cell in the grid and the contour: f(x , y)=c



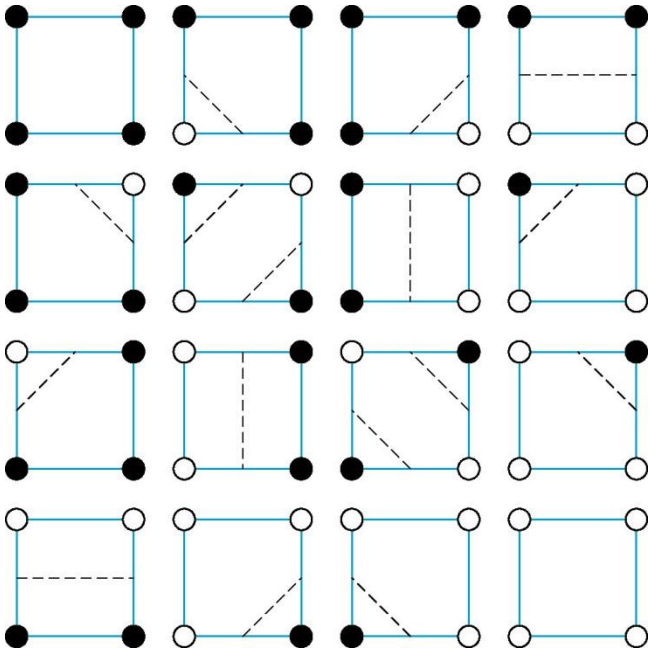
For these conditions, we have the possibilities:

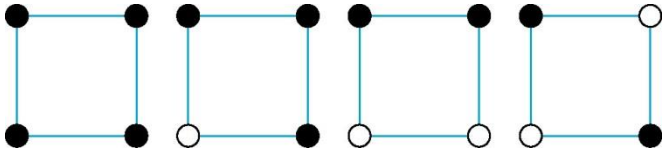


We approximate or interpolate based on the values of f(x , y) at the vertices assuming single intersection

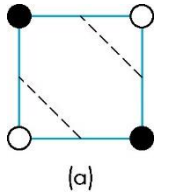
Intersection possibilities

There are 16 (2^4) way to color the vertices in black (less than c) and white (greater than c). From these 16 there are only 4 unique cases. Each of the other cases is symmetric with one of these 4

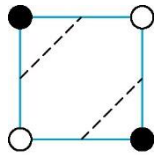




What we need now is how to draw a line segment inside each cell (if one exist) to approximate the contour. The line segment could be unambiguously approximated except in the last case which suggests two ways to put the two line segments (a, b). Which pair we choose: Random choice, Always use one of them, or subdivide the cell to remove ambiguity



(a)



(b)